# Growing Neural Gas for Time Series Evaluation

ADVANCED SEMINAR

submitted by
Jan Ehrensperger

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

Prof. Dr Jörg Conradt

Supervisor: Dipl.-Inf. Nicolai Waniek
Final Submission: 15.10.2015

In your final hardback copy, replace this page with the signed exercise sheet.

**Abstract**

This thesis will describe the following eight *prototype-based* clustering algorithms: SOM, NG, ENG, MNG, GCS, GNG, RGNG, MGNG. Thereto the basic principles of the algorithms functionality are explained and the a priori defined parameters are indicated. After this explanation their field of application are presented and a flow sheet of the functional specialisation/generalisation among the given algorithms is exhibited.

# Contents

# 1 Introduction

The purpose of cluster analysis is the information retrieval (data-mining) of a huge amount of data. Therefore clustering-algorithms are splitting the data into clusters. Within a cluster, the data features on a high degree of similarity. The aim is to find new clusters (patterns) for classification and system prediction after a finite time of processing. In general clustering-algorithms map a set of $N$ input vectors: $\mathbf{X} = \{x_1, x_2, \ldots, x_n | \forall x_i \in \mathbb{R}^d, i = 1, 2 \ldots, N\}$ into $c$-clusters. Whereby $2 \leq c \leq N$.
There is no standardisation to classify cluster algorithms. Traditionally, these algorithms are divided into hierarchical- and partitioning-clustering. Clusters generated by hierarchical clustering methods are presented in a dendrogram (hierarchic structure). Partitioning cluster methods subdivide the input space into clusters, depending on density functions or other similarities.
In this thesis, all algorithms belong to the so called *prototype-based clustering* (PBC). These methods refer more easily to hierarchical clustering methods. In PBC each input vector is represented by a single point $\mathbf{X}$ (prototype) in the input space. A high concentration of multiple prototypes within a given region is called a cluster. The aim of this thesis is to reveal how PBC-algorithms work especially those using artificial neural networks (see following section).

# 2 An overview of selected artificial neural network for cluster analysis

In this chapter an overview on selected algorithms, using artificial neural networks is presented. Therefore the functional principle of each algorithm will be explained and afterwards a priori defined parameters are specified. If some field of application or some performance measurements are mentioned in the references, they will be described in this thesis. However the focus of this thesis is placed on the evaluation of how far the algorithms are suitable for time series evaluation.

## 2.1 Self-Organizing Map (SOM)

Self-organizing maps (SOM) were proposed in the 1980s. They represent an unsupervised learning algorithm, where the output classification is generated without external help. During learning, the competitive learning rule is used, in which only the best fitting neuron (*winner-takes-all neuron*) will be activated. The SOM-algorithm is based on the knowledge that the human brain maps sensations into a linear or planar topology. Therefore a discrete two-dimensional grid, called *competitive layer* will be generated.

The competitive learning rule can be observed in human brains. Active neurons are suppressing the activity of other neuron-groups in their direct neighbourhood. As a consequence, the action-potential of a neuron can be computed by summing up the suppression of the neighbourhood and the adaptation concerning the input vector.

**How does self-organising work:**
Mainly four components are required:

1. **Initialisation**: At the beginning, each node in the discrete competitive layer is connected with the input space via a weighted connection. Assuming the input space is $D$-dimensional, the input vectors can be written as $\mathbf{x} = \{x_i : i = 1, \ldots, D\}$ and the weighted connection $\mathbf{w_j}$ between input space and competitive layer as $\mathbf{w_j} = \{w_{ji} : j = 1, \ldots, N; i = 1, \ldots, D\}$, whereby $N$ represents the total number of neurons of the competitive layer. During initialisation $\mathbf{x}, \mathbf{w_j}$ are assigned randomly.

2. **Competition:** For each neuron in the competitive layer exists a relation between input space and weight-vector according to a discriminant function $d(\cdot)$, e.g. $d(\cdot)$ as euclidean distance between the vectors given as: $d_j(\mathbf{x}) = \sum_{i=1}^{D}(\mathbf{x}_i - \mathbf{w}_{ji})^2$. The neuron with the smallest value $d_j$ is called *winner-takes-all* neuron.

3. **Cooperation process:** Neurons within a topological neighbourhood of an active neuron could be switched on during the next iteration. This excitation diminishes with increasing distance to the winner-takes-all neuron. Therefore $S_{ij}$ describes the topological distance between neurons within the competitive layer. The position of the winner-takes-all neuron is given as $I(\mathbf{x})$. The excitation of the neighbourhood could be computed as $T_{j,I(\mathbf{x})} = exp(\frac{-S^2_{j,I(\mathbf{x})}}{2\sigma^2})$ whereby $\sigma(t) = \sigma_0 exp(\frac{-t}{\tau_\sigma})$. $\sigma(t)$ describes the reduction of excitation during progression of time.

4. **Adaptation process:** The self-organisation process will be established by adjusting the weights of each neuron. Therefore the new weight-vectors are calculated as $\Delta\mathbf{w}_{ji} = \nu(t) \cdot T_{j,I(\mathbf{x})}(t) \cdot (\mathbf{x}_i - \mathbf{w}_{ji})$. With $\nu(t)$ as learning-rate, which is defined by: $\nu(t) = \nu_0 exp(\frac{t}{\tau_\nu})$. The winner-takes-all neuron and its direct topological neighbourhood will move towards the input vector by adjusting their weight-vectors. The competitive layer will converge to its representation of the input manifold.

**How does the SOM algorithm work:**
The steps of the algorithm can be derived from the steps of the self-organisation which leads to the following algorithm summary:

1. **Initialisation**: Assign random values for each weight-vector $\mathbf{w}_j$.
2. **Sampling**: Draw an input vector $\mathbf{x}$ from set of trainings data of the $D$-dimensional input space.
3. **Matching**: Find the winner-takes-all neuron within the competitive layer.
4. **Updating**: Compute the new weight vector for each neuron.
5. **Continuation**: Repeat from step 2 until the weight adjustment is smaller than a threshold.

**A priori defined parameters:**
The time invariant parameters $\sigma_0, \tau_\sigma, \nu_0, \tau_\nu$ have to defined beforehand. The iteration termination can either be a fixed number of iterations or a residuum which fulfil the following condition between two iteration-steps: $\Delta\mathbf{w}_{ji}(t+1) - \Delta\mathbf{w}_{ji}(t) < \epsilon; \forall\epsilon > 0$. Another parameter is the total number of neurons within the competitive layer, which have a big impact on the result of the clustering analysis. The SOM-algorithm forms the basis for the following PBC-algorithms.

**Field of application:**
[MBS93, p.2] describe that the SOM-algorithm was successfully installed in speech recognition, image processing and robot control.

## 2.2 Neural Gas (NG)

The *Neural Gas* (NG) algorithm was presented in 1991 and is based on the SOM-algorithm. The aim are vector quantisation respectively data compression. The neurons are not arranged in a fixed topological ordering. All neurons are able to move freely and have the ability to establish a connection towards neurons in their neighbourhood. The NG-algorithm is trained via an unsupervised learning, too. In comparison to SOM, it converges faster and avoids local extrema by using a kind of simulated annealing.

**How does the NG-algorithm work:**
The algorithm consists of the following seven steps:

0. **Initialisation:** Create $N$-neurons by creating $N$ weight vectors $\mathbf{w}$ whereby $\mathbf{w}_i \in \mathbb{R}^n$. Also create a connection matrix $C$ where $C_{ij} = 0$, which mean that there is no connection between two neurons $i$ and $j$. $C_{ij} = 1$ connotes a connection between the neurons.

1. **Sampling:** Draw an input vector $\mathbf{v}$ from input manifold $M$, whereby $M \subseteq \mathbb{R}^n$ and $M$ for example can be a geometric figure.

2. **Matching:** Divide up the input manifold $M$ into the so called *voronoi polyhedra*. With $M_i = \{\mathbf{v} \in M | \; ||\mathbf{v} - w_i|| \leq ||\mathbf{v} - \mathbf{w}_j|| \forall j\}$. The sequence of neurons $(i_0, i_1, \ldots, i_{N-1})$ is determined for example by $||\mathbf{v} - \mathbf{w}_{i_0}|| \leq ||\mathbf{v} - \mathbf{w}_{i_1}|| \leq \cdots \leq ||\mathbf{v} - \mathbf{w}_{i_{N-1}}||$. Any input vector has a certain degree of influence on each neuron, depending on their euclidean distance. Therefore the learning rule is known as *winner-take-most*.

3. **Adaptation:** Compute the new weight vectors with: $\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \epsilon \cdot \exp\left(\frac{-k_i}{\lambda}\right)(\mathbf{v} - \mathbf{w}_i^{old}), i = 1, \ldots, N$ [MS91]. This leads to [MBS93, equation 7] by transformation:$\Delta \mathbf{w}_i = \epsilon \cdot h_\lambda(k_i(\mathbf{v}, \mathbf{w})) \cdot (\mathbf{v} - \mathbf{w}_i)$, whereby $h_\lambda(k_i(\mathbf{v}, \mathbf{w})) = \exp\left(\frac{-k_i(\mathbf{v}, \mathbf{w})}{\lambda}\right)$. By setting $\lambda \to 0$ leads to the adaptation rate of the *k-means*-clustering algorithm. $\lambda \neq 0$ results in the implementation of the *winner-takes-most* learning rule. In general $\lambda$ stands for the decay constant and $\epsilon$ for the step size.

4. **Connection:** If there was no connection between two neurons ($C_{i_0 i_1} = 0$) create it ($C_{i_0 i_1} = 1$) and set its age value to zero ($t_{i_0 i_1} = 0$). If two neurons were connected before, reset its age back to zero.

5. **Ageing:** Increase the age for each emanating edge of the winner neuron by one ($t_{i_0 j} = t_{i_0 j} + 1$).

6. **Selection:** Remove all connection of $i_0$ in respect of $C_{i_0 j} = 1 \wedge t_{i_0 j} > T \Rightarrow C_{i_0 j} = 0$, whereby $T$ represents the life-time of a connection. Continue with step one.

**A priori defined parameters:**
First of all, a number of neurons $N$ and a maximal number of iterations $t_{max}$ shall be determined. [MBS93] showed that these parameters can be determined by minimisation of a cost function (kind of distortion error).
The decay constant $\lambda$, step size $\epsilon$ and the life-time $T$ can computed with $g(t) = g_i(g_f/g_i)^{\frac{t}{t_{max}}}$, where $t$ is the current iteration. In [MS91] the following parameters were used: $\lambda_i = 30, \lambda_f = 0.01$, $\epsilon_i = 0.3, \epsilon_f = 0.05$, $T_i = 20, T_f = 200$. In general $\lambda_i, \lambda_f, \epsilon_i, \epsilon_f, T_i, T_f$ could be optimized for a special problem by using a comprehensive parameter study.

**Performance:**
In the second step of the algorithm all values are sorted, which consumes the most processing time. This leads to a complexity of $O(Nlog_2N)$ for sequential processing.

**Fields of application:**
Beside vector quantisation, data compression, speech recognition, image processing or pattern recognition the NG-algorithm was successfully installed in function approximation ([MBS93, Kap.6]) of the so called *Mackey-Glass* differential equation with prediction of their development over time. For time series prediction some additional parameters are established which have to be initialised. Therefore the out of the box NG-algorithm could not be used for time series evaluation.

### 2.2.1 Enhanced Neural Gas (ENG)

The *Enhanced Neural Gas* (ENG)-algorithms was proposed in [QS05] in 2005. ENG extends NG by make it more stable due to input sorting, position of outlier and parameter selection. The algorithm is also influenced by the *Robust Growing Neural Gas* algorithm (see section 2.4.1).

**How does the ENG-algorithm work:**
The ENG-algorithm is based on NG and consists as well as NG of seven steps. [QS05, p.1282] is the basis for the following pseudo code.

0. **Initialisation:** Create a set of weight vectors $W = \{w_1, w_2, \ldots, w_N\}$ whereby each weight vector is initialized randomly. Also create an iteration variable $t$ and initialise it: $t = 1$. Thereby the total number of iteration can be computed as $iter = m \cdot N + t$, with $m$ as trainings epoch and $N$ as number of neurons.

1. **Sampling:** Compute for each weight vector the average harmonic mean distance regarding its position in input space : $d_m^i(0) = \left[\frac{1}{N}\sum_{j=1}^{N}\frac{1}{||\mathbf{v}_j - \mathbf{w}_i^{m\cdot N}||}\right]^{-1}$; $i = 1, 2, \ldots, N$

2. **Matching:** Draw an input vector, create the voronoi polyhedra with $||\mathbf{v}_t^m - \mathbf{w}_i^{iter}||$ and sort it descendent.

3. **Adaption:** Compute the new weight vectors with $\Delta\mathbf{w}_i = \epsilon(iter) \cdot h_\lambda(k_i(\mathbf{v}, \mathbf{w})) \cdot exp\left(-\frac{||\mathbf{v} - \mathbf{w}_i||}{\beta \cdot d_i^m(0)}\right) \cdot \sigma_i(iter) \cdot \frac{(\mathbf{v} - \mathbf{w}_i)}{||\mathbf{v} - \mathbf{w}_i||}; i = 1, 2, \ldots, N$. Whereby the parameters $\epsilon, \lambda, \beta$ are determined as:

   - $\epsilon(iter) = \epsilon_i(\frac{\epsilon_f}{\epsilon_i})^{\frac{iter}{m_{max} \cdot N}}$
   - $\lambda(iter) = \lambda_i(\frac{\lambda_f}{\lambda_i})^{\frac{iter}{m_{max} \cdot N}}$

- $\beta(iter) = \begin{cases} \beta_i(\frac{\beta_m}{\beta_i})^{\frac{iter}{(m_{max}-m_{ad}-1)\cdot N}}; & m < m_{max} - m_{ad} \\ \beta_m(\frac{\beta_f}{\beta_m})^{\frac{iter-(m_{max}-m_{ad})\cdot N-1}{m_{ad}\cdot N}}; & m \geq m_{max} - m_{ad} \end{cases}$,
  with $m_{ad}$ as iteration number in which the topology could change.

4. **Connection:** Determine the two neurons with lowest weight vector: $s_1 = \arg\min_{\mathbf{w}_i \in W} ||\mathbf{v}_j - \mathbf{w}_i^{iter}|| \wedge s_2 = \arg\min_{\mathbf{w}_i \in W \setminus \{s_1\}} ||\mathbf{v}_j - \mathbf{w}_i^{iter}||$. If a connection exists, reset its age, else create a new connection.

5. **Recombination:** Find the neuron with the greatest local error regarding to an error function ($\mathbf{w}_f$). Then determine the neuron with the greatest error within the direct topological neighbourhood ($\mathbf{w}_q$). Create a new neuron with its weight vector $\mathbf{w}_r = \frac{2 \cdot \mathbf{w}_q + \mathbf{w}_f}{3}$. Remove the existing connection between $f$ and $q$. Afterwards create a new connection among $f, r, q$.

6. **Ageing:** Increase the age counter of each unaffected connection.

7. **Selection:** Remove all connections whose age is over maximum life-time. Continue with step one until $m_{max}$ or a predefined performance criteria is reached.

**A priori defined parameters:**
The ENG algorithm require the definition of the following parameters: $\epsilon_i, \epsilon_f, \lambda_i, \lambda_f, \beta_i, \beta_m, \beta_f, \kappa, \eta$. Where $\kappa, \eta$ are needed to determine the neuron with largest error. [QS05, p.1281] initialised those parameters with the following values: $\epsilon_i = 0.8, \epsilon_f = 0.05, \lambda_i = 10, \lambda_f = 0.01, \beta_i = 50, \beta_m = 10, \beta_f = 0.01, \kappa = 2, \eta = 1e^{-4}$. Furthermore $m_{max} = 10 \wedge m_{ad} = 5$ are chosen.

**Field of application:**
The ENG-algorithm is more robust towards noisy input vectors, because it inserts new neurons in highly error-prone regions. By inserting new neurons the degree of freedom of the system is increased and therefore a better adaptation (see Growing Neural Gas algorithm) will be achieved. ENG could be used to find compact cluster ([QS05, p.1286]), but it might not find loosely clusters ([QS05, p.1286]).

### 2.2.2 Merge Neural Gas (MNG)

The *Merge Neural Gas* (MNG)-algorithm is a synthesis of the *Merge Self Organizing Map* (MSOM-[HMN+05]) and NG. MNG create a recursive data model, which is able to process time series like DNA-chains or articulation within speech recognition. It resembles the NG-algorithm, but additionally needs a *context vector*. The best fitting neuron of previous iteration step $I_{n-1}$ is determined as a weighted linear combination

of weight- and context vector (merge). During current iteration, the context vector describes the target, which should be achieved by adaptation of the winner $I_n$ neuron and its topological neighbourhood.

**How does the MNG-algorithm work:**
The algorithm is similar to NG, but instead of connection information among the neurons, a context vector for each neuron is required.

0. **Initialisation:** Create for each of $N$-neurons a tuple consisting of a weight vector $\mathbf{w}_i$ and a context vector $\mathbf{c}_i$, whereby: $n_i(\mathbf{w}_i, \mathbf{c}_i) \in \mathbb{R}^d \times \mathbb{R}^d$, with $d$ as dimension of the input space. The values of the tuples vectors are initialised randomly.

1. **Sampling & Matching:** Get an input vector $\mathbf{v}$ from input manifold $M$. Determine the distance given as: $d_i(n) = (1-\alpha) \cdot ||\mathbf{v} - \mathbf{w}_i||^2 + \alpha \cdot ||\mathbf{c}(n) - \mathbf{c}_i||^2$, with $\alpha$ as balancing parameter between $\mathbf{w}_i$ and $\mathbf{c}_i$. For the current context $\mathbf{c}(n) = (1-\beta) \cdot \mathbf{w}_{I_{n-1}} + \beta \cdot \mathbf{c}_{I_{n-1}}$. takes effect. $I_{n-1}$ is the winner neuron of the previous iteration and $\beta$ represents the degree of merging the context within two frames

2. **Adaptation:** Compute the new weight- and context vector, with:

   - $\Delta\mathbf{w}_i = \epsilon_w(k) \cdot h_{\lambda(k)}(r(d_i, d)) \cdot (\mathbf{v} - \mathbf{w}_i)$, whereby $r(d_i, d)$ describes the rank of the $i$-th neuron upon the current input-vector $\mathbf{v}$ and the context representation $d$.
   - $\Delta\mathbf{c}_i = \epsilon_w(k) \cdot h_{\lambda(k)}(r(d_i, d)) \cdot (\mathbf{c}(n) - \mathbf{c}_i)$, with $r(d_i, d)$ as described above.

   $h_{\lambda(k)}(r(d_i, d))$ represents the topological neighbourhood ranking function, which is defined as: $h_{\lambda(k)}(r(d_i, d)) = exp\left(\frac{-r(d_i, d)}{\lambda(k)}\right)$. $\lambda(k)$ is responsible for the influence of the neighbourhood and can be computed with $\lambda(k) = \lambda_0 \cdot \left(\frac{\lambda_f}{\lambda_0}\right)^{\frac{k}{k_{max}}}$. Continue with step one, until the termination condition is reached.

**A priori defined parameters:**
Beside the NG-algorithm parameters $\alpha, \beta$ have to be determined. Whereby $\alpha$ describes the balancing between weight and context vector, which is important to determine the winner neuron. In [HMN+05, S.47] $\alpha$ is given as $\alpha < 0.5$, because the adaptation towards the current pattern should have an higher influence, than towards the context. $0 \leq \beta \leq 1$ describe the influence of the past context vectors towards the current one. In [HMN+05, S.7] $\beta$ is given as $\beta = 0.5$.

**Field of application:**
The MNG-algorithm was successfully installed in medical research for recognition of *sleep spindle* in electroencephalography (EEG)-recordings ([EZH+07]). In [EZRZ07] the MNG-algorithm is transformed into the *Window Merge Neural Gas*-algorithm by defining the context as a window of the past $\zeta$ signals.

## 2.3 Growing Cell Structures (GCS)

The *Growing Cell Structures* (GCS) algorithm is based on NG and was presented as supervised as well as unsupervised learning algorithm in 1994. It is geared towards a bijection between input and output space, in which closely spaced input vectors activates closely spaced neurons and the other way round. In particular, input vectors with a high probability of occurrence yield a finer discretisation.
The algorithm begins with a $k$-dimensional simplex. A simplex for $k = 1$ is a line, $k = 2$ a triangle and $k = n$ a n-dimensional hyper-tetrahedron. The network of neurons will always be structured in $k$-dimensional simplexes.

**How does the GCS-algorithm work:**
The professional basis for the following pseudo-code is [HA00, p.209] and [Fri93, p.1442ff].

0. **Initialisation:** Start with **one** $k$-dimensional randomly placed simplex. Each vertex represents a neuron consisting of a randomly initialised weight vector $\mathbf{w}_{c_i} \in \mathbb{R}^n$ and a winner counter $E$. The winner counter counts how often the referring neuron fits best on input vector. $k < n$ will lead to an dimensionality reduction and the edges of the simplexes represents the neighbourly relations among the neurons.

1. **Sampling & Matching:** Draw an input vector $\zeta$ and compute the winner neuron $bmu$ with the smallest euclidean distance towards $\zeta$:
$bmu = ||\zeta - w_c||_{min_{c \in network}}$. Increment the winner counter $E$ of $bmu$-neuron.

2. **Adaptation:** Adjust the weight vector for $bmu$ and its topological neighbours with: $\Delta\mathbf{w}_{bmu} = \epsilon_{bmu}(\zeta - \mathbf{w}_{bmu})$ and $\Delta\mathbf{w}_n = \epsilon_i(\zeta - \mathbf{w}_n)(\forall n \in \text{neighbourhood})$. This results in a movement of the winner unit and its neighbours towards the input vector by a user defined adaptation parameter.

3. **Recombination:** After processing $\Xi$-input vectors a new neuron ($w_{new}$) is created between the neuron with the highest winner counter ($w_{bmu}$) and its furthest neighbour ($w_f$), whereby $w_{new} = \dfrac{(w_{bmu} + w_f)}{2}$. Afterwards, the hyper-tetrahedron configuration has to maintained by creating adequate connection among $w_{new}, w_{bmu}$ and $w_f$. The winner counter $E_{new}$ is created from parts of a fraction of its topological neighbours with
$E_{new} = \sum \left( \dfrac{1}{|n|} E_n, \quad \forall n \in \text{neighbourhood of } w_{new} \right)$.

4. **Selection:** Every $\Lambda$-iteration the neuron with the greatest average euclidean distance will be deleted ($max_{c \in \text{network}} = \left( \frac{\sum ||\mathbf{w}_c - \mathbf{w}_n||}{card(n)}; \forall n \in network \right)$) and all neurons, which are not located in a hyper-tetrahedron configuration, with it.

5. **Ageing:** Decrease the winner counter $E$ of each Neuron with $\Delta E_c = -\beta E_c; \forall c \in network$ for a simulated decay process.
   Continue with the first step until a termination condition is reached.

**A priori defined parameters:**
$\epsilon_{bmu}$ and $\epsilon_n$ describe the adaptation of the winner neuron and its topological neighbours towards the input vector. $\Xi, \Lambda$ describe after how many iterations a new neuron is created/deleted. For the decay process the constant $\beta$ has to be determined.

## 2.4 Growing Neural Gas (GNG)

The *Growing Neural Gas* (GNG)-algorithm is a generalisation of the GCS-algorithm and based on NG, too. The neurons are able to engender any topology as opposed to GCS and in contrast to NG it does not need any dynamically modifiable parameters. In 1995, the algorithm was presented and uses the *hebbian learning*, too.

**How does the GNG-algorithm work:**
[Fri95, p.4ff] was used for the following pseudo-code:

0. **Initialisation:** Create two neurons $a$ and $b$ with related randomly initialised weight vectors $\mathbf{w}_a$ and $\mathbf{w}_b \in \mathbb{R}^n$. Also matrices are need to store connection information and connection age information. In addition each neuron has a counter (error counter), representing the cumulated error.

1. **Sampling & Matching:** Draw an input vector $\xi$ from set of input vectors and locate the two neurons with the minimal distance toward $\xi$ with:
   $s_1 = \min_{\mathbf{w}_i \in N}(||\xi - \mathbf{w}_i||) \wedge s_2 = \min_{\mathbf{w}_i \in N \setminus \{s_1\}}(||\xi - \mathbf{w}_i||)$. Increment the age of all connection emanating $s_1$. Also sum up the error counter with $\Delta error(s_1) = ||\mathbf{w}_{s_1} - \xi||^2$.

2. **Adaptation:** Compute the displacement of $s_1$ and its direct topological neighbours in the direction of $\xi$ with $\Delta \mathbf{w}_{s_1} = \epsilon_b(\xi - \mathbf{w}_{s_1}); \Delta \mathbf{w}_n = \epsilon_n(\xi - \mathbf{w}_n)$. Whereby $\epsilon_b \, \epsilon_n \in (0, 1]$ represents parameters for adjusting the movement.

3. **Selection:** Reset the connections age, if $s_1$ and $s_2$ were connected, otherwise create one. Delete all connections with an age higher than the predefined maximal age $a_{max}$. Also delete neurons which have no emanating connections. Increment the age of all emanating connection of $s_1$ by one.

4. **Recombination:** Every $\lambda$-iteration, locate the neuron $q$ with the greatest value of its error counter and its topological neighbour with the highest error counter $f$. Create a new neuron $r$ between $q$ and $f$ with $\mathbf{w}_r = \frac{\mathbf{w}_q + \mathbf{w}_f}{2}$. Also create connections among $r$, $q$ and $f$ and delete the original connection between $q$ and $f$.
   Reduce the error counter of $q$ and $f$ by a multiplication with $\alpha$. Initialise the error counter of the new neuron with the value of $f$.

5. **Ageing:** Reduce the error counter of all neurons by multiplication with $\delta$ and continue with step one.

**A priori defined parameters:**
In contrast to NG the parameters are not modified dynamically. The following parameters have to be determined before launching the algorithm: $N, \epsilon_b, \epsilon_n, a_{max}, \lambda, \alpha, \delta$. $N$ represents the maximal number of neurons, $\epsilon_b, \epsilon_n$ describe the mobility of the winner neuron and its neighbours. [Fri95] used $\epsilon_b = 0.2$ and $\epsilon_n = 0.006$. The parameter $a_{max}$ describes the maximal age of a connection and every $\lambda$ iteration a new neuron will be created. $\alpha$ represents the reduction of error counter by inserting a new neuron and $\delta$ will reduce the overall value of the error counter every iteration step.

### 2.4.1 Robust Growing Neural Gas (RGNG)

The *Robust Growing Neural Gas* (RGNG)-algorithm was presented in 2004 and should improve the robustness towards input sorting, initialisation and position of outliers. Furthermore the RGNG determines dynamically the optimal number of neurons during runtime. Therefore the principle of the *Minimum Description Length* (MDL) is used. Thereby the RGNG-network with the global minimum of the MDL is searched, which represents the optimal encoding strategy for the given problem. The RGNG enhances GNG and is also an unsupervised learning algorithm.

**How does the RGNG-algorithm work:**
[QS04] is the basis for the following pseudo-code. The authors reused many parts of the RGNG in the ENG-algorithm.

0. **Initialisation:** Like the initialisation of the GNG-algorithm, the RGNG starts with two neurons as well, given as a tuple of weight- and context vector: $(\mathbf{w}_i, \mathbf{c}_i)$. The control variable is given as $t = 0$, whereby the total iteration number can be computed with $iter = m \cdot N + t$. $m$ is the current training epoch and $N$ is the actual number of neurons.

1. **Sampling & Matching:** Draw an input vector $\mathbf{v}$. Compute the harmonic distance of each neuron, according to its position with:
   $d_m^i(0) = \left[ \frac{1}{N} \sum_{j=1}^{N} \frac{1}{||\mathbf{v}_j - \mathbf{w}_i^{m \cdot N}||} \right]^{-1} ; i = 1, 2, \ldots, N$

Also compute the new adaptation rate with: $\epsilon_b^l = \epsilon_{bi}^l(\frac{\epsilon_{bf}^l}{\epsilon_{bi}^l})^{\frac{prenode^l}{pre\_numnode}}$ and $\epsilon_n^l = \epsilon_{ni}^l(\frac{\epsilon_{nf}^l}{\epsilon_{ni}^l})^{\frac{prenode^l}{pre\_numnode}}$. Where $\epsilon_b^l$ is the learning rate of the winner and $\epsilon_n^l$ represents the learning rate of its topological neighbours, whereby $l = 1, \ldots, c$. Initial and final values of $\epsilon_b^l, \epsilon_n^l$ are $\epsilon_{nf}^l, \epsilon_{ni}^l, \epsilon_{bf}^l, \epsilon_{bi}^l$ with $c$ as the current number of neurons and $pre\_numnode^l$ as the ranking index, which is zero for newly inserted neurons and one if a newer neuron is inserted. Afterwards increase the control variable $t = 1$ and $iter = m \cdot N + t$ and locate the two neurons $s_1$ and $s_2$ with the smallest euclidean distance towards the input vector.

2. **Adaptation:** Compute the new weight vector for the winner neuron with: $\Delta\mathbf{w}_{s_1} = \epsilon_b^{s_1}\sigma_{s_1}(iter)\frac{(\mathbf{v}-\mathbf{w}_{s_1})}{||\mathbf{v}-\mathbf{w}_{s_1}||}$ and for its topological neighbours: $\Delta\mathbf{w}_i = \epsilon_n^i\sigma_i(iter)\frac{(\mathbf{v}-\mathbf{w}_i)}{||\mathbf{v}-\mathbf{w}_1||} + \exp\left(-\frac{d_{s_1i}}{\zeta}\right)\cdot\beta\frac{\sum_i d_{s_1i}}{|N_{s_1}|}\frac{(\mathbf{v}-\mathbf{w}_{s_1})}{||\mathbf{v}-\mathbf{w}_{s_1}||}$. Whereby the parameter are determined with:

   - $\sigma_k(iter) = \sigma_k^m(t) = \begin{cases} d_k^m(t); & ||\mathbf{v}_t^m - \mathbf{w}_k^{iter}|| \geq d_k^m(t-1) \\ ||\mathbf{v}_t^m - \mathbf{w}_k^{iter}||; & ||\mathbf{v}_t^m - \mathbf{w}_k^{iter}|| < d_k^m(t-1) \end{cases}$
   - $d_k^m(t) = \begin{cases} \left\{\frac{1}{2}\left[\frac{1}{d_k^m(t-1)} + \frac{1}{||\mathbf{v}_t^m - \mathbf{w}_k^{iter}||}\right]\right\}^{-1}; & ||\mathbf{v}_t^m - \mathbf{w}_k^{iter}|| \geq d_k^m(t-1) \\ \frac{1}{2}\left[d_k^m(t-1) + ||\mathbf{v}_t^m - \mathbf{w}_k^{iter}||\right]; & ||\mathbf{v}_t^m - \mathbf{w}_k^{iter}|| < d_k^m(t-1) \end{cases}$
     with $d_k^m(0) = \left[\frac{1}{N}\sum_{j=1}^N \frac{1}{||\mathbf{v}_j - \mathbf{w}_i^{m\cdot N}||}\right]^{-1}; i = 1, 2, \ldots, N$

3. **Selection:** Reset the age of the connection between $s_1$ and $s_2$ if it existed before, otherwise create it. Than delete all connections with a higher age than $a_{max}$. If a neuron has no emanating edges any more, delete it too.

4. **Recombination:** Ascertain the MDL-value for the actual configuration (see [QS04, S.1142ff]). If the value is smaller than the previous configuration, save the current one, because it is a better network representation of the given problem. Locate the neuron with greatest local error $w_q$ with: $\sum_{x\in S_i}\exp\left(-\frac{||\mathbf{x}-\mathbf{w}_i||}{\left[\frac{1}{|S_i|}\sum_{x\in S_i}\frac{1}{||\mathbf{x}-\mathbf{w}_i||}\right]^{-1}}\right)||\mathbf{x}-\mathbf{w}_i||$, whereby $S_i$ is the exposure field of the neuron which is represented by $\mathbf{w}_i$. Also locate the neuron $\mathbf{w}_f$ among the topological neighbourhood of $\mathbf{w}_q$ with the greatest error.

Afterwards create a new neuron weight with $\mathbf{w}_r = 2 \cdot \mathbf{w}_q + \frac{\mathbf{w}_f}{3}$ and replace the connection between $\mathbf{w}_q$ and $\mathbf{w}_f$ with connections among $\mathbf{w}_q, \mathbf{w}_f, \mathbf{w}_r$.
Than put $pre\_numnode^r = 0$, $pre\_numnode^l : += 1$ and continue with step one.

**A priori defined parameters:**
The additional robustness is bought dearly with additional parameter. Therefore the following parameters have to be determined: $\epsilon_{bi}, \epsilon_{bf}, \epsilon_{ni}, \epsilon_{nf}, \alpha_{max}, \beta, \kappa, \eta$. Where $\epsilon_{bi}, \epsilon_{bf}, \epsilon_{ni}, \epsilon_{nf}$ represents the limit of the learning rate, finding new clusters. $\alpha_{max}$ is the maximal age of a connection and $\beta$ represents the mobility of the winners neighbourhood towards the input vector. $\kappa, \eta$ are needed to determine the MDL value.

### 2.4.2 Merge Growing Neural Gas (MGNG)

The *Merge Growing Neural Gas* (MGNG)- algorithm was proposed in 2009 and is a combination of MNG and GNG. Therefore it suits for time series analysis. During the learning phase, the configuration's entropy is maximised by inserting new neurons.

**How does the MGNG-algorithm work:**
The MGNG algorithm is presented hereafter, it bases mainly on GNG but also uses some features of the MNG-algorithm ([AvHHB09, p.5ff]).

0. **Initialisation:** Create, like the initialisation of GNG, two unconnected neurons with corresponding randomly initialised weight vector $\mathbf{w}_i$, context vector $\mathbf{c}_i$ and a counter variable $e_i = 0$. The global context $\mathbf{C}$ is initialised with zero.

1. **Sampling & Matching:** Draw an input vector $\mathbf{x}_t$ and locate the winner neuron $r$ and the second best $s$, where $r = \arg\min_n d_n(t)$ and $s = \arg\min_{n \setminus \{r\}} d_n(t)$, with $d_n(t) = (1-\alpha)||\mathbf{x}_t - \mathbf{w}_i||^2 + \alpha||\mathbf{C}_t - \mathbf{c}_i||^2$.
The global context can be computed with $\mathbf{C}_{t+1} = (1-\beta) \cdot \mathbf{w}_r + \beta \cdot \mathbf{c}_r$. Afterwards increment the age of all edges and the counter variable of the winner $e_r$ by one.

2. **Adaptation:** Compute the movement within input space of the winner $r$ and its neighbourhood neurons by adjusting the weight vectors by following equations:

   - The best matching neuron:$\mathbf{w}_r = \mathbf{w}_r + \epsilon_b(\mathbf{x}_t - \mathbf{w}_r)$ and $\mathbf{c}_r = \mathbf{c}_r + \epsilon_b(\mathbf{C}_t - \mathbf{c}_r)$.
   - The topological neighbours of the winner: $\mathbf{w}_n = \mathbf{w}_n + \epsilon_n(\mathbf{x}_t - \mathbf{w}_i)$ and $\mathbf{c}_n = \mathbf{c}_n + \epsilon_n(\mathbf{C}_t - \mathbf{c}_i)$.

3. **Selection:** Reset the age of the connection between $r$ and $s$ if it exists beforehand, otherwise create a connection. Afterwards delete all edges with an age greater or equal $\theta$. In addition, delete all neurons, not having any emanating edges.

4. **Recombination:** Every $\lambda$-iteration create a new neuron $l$. Therefore locate the neuron $q$ with the highest counter variable $q = \arg\max_n e_n$ and the neuron $f$ with the highest counter among $q$'s neighbours. The weight vector of the new neuron can be determined with $\mathbf{w}_l = 0.5(\mathbf{w}_q + \mathbf{w}_f)$; $\mathbf{c}_l = 0.5(\mathbf{c}_q + \mathbf{c}_f)$ and $e_l = \delta \cdot (e_f + e_q)$. Replace the original connection between $f$ and $q$ with two connections $f, l$ and $l, q$. Reduce the value of the counter variables of $q$ and $f$ with: $e_q \vee e_f : \cdot = (1 - \delta)$

5. **Ageing:** Reduce the value of every counter variable with : $e_n = \eta \cdot e_n$. Continue with step one until a maximal number of neurons or a performance criteria is reached.

**A priori defined parameters:**
Either define a maximal number of neurons or a performance criteria for terminating the iterations and determine following parameters: $\alpha, \theta, \beta, \lambda, \delta, \eta, \epsilon_n, \epsilon_b$. Whereby $\alpha \in [0, 1]$ represents the importance of the current towards the past signals. $\beta \in [0, 1]$ describes the influence of the past calculations towards the current one. $\theta$ is the maximal age of a connection. Furthermore, $\lambda$ describes after how many iterations a new neuron is established. The parameter $\delta$ changes the activation of the neurons and $\eta$ is responsible for weighting current transformations higher than the past ones. [AvHHB09] determined the parameters as following: $\alpha = 0.5, \theta = 100, \beta = 0.75, \lambda = 600, \delta = 0.5, \eta = 0.9995, \epsilon_n = 0.0006, \epsilon_b = 0.05$.

# 3 Dependencies and functional specialisation among the algorithms

After presenting selected *PBC*-algorithms, this section shall clarify some dependencies among the algorithms and build a kind of *dependency tree* for the algorithms temporal- as well as functional-evolution. For illustration see figure 1 in which indirect relations (e.g.: relations of the parent's parent) are not explicitly shown.
The in figure 1 depicted *Self-Organizing Maps* (SOM) algorithm was proposed in the 80th and represents the basis for the other algorithms. In 1991 the *Neural Gas* (NG) algorithm generalised the SOM insofar that an a priori defined number of neurons are not arranged in a fix topology and are able to create/delete connections. Some years later (in 1994) he *Growing Cell Structures* (GCS)-algorithm was proposed. Whereby GCS is a specialisation of SOM on the one hand because the neurons are arranged in a fix topology (simplexes) and a generalisation of NG on the other hand due to neuron creation/deletion capability. Based on GCS, the generalisation in terms of the *Growing Neural Gas* (GNG) algorithm was proposed in 1995. As a result the the fixed topology of GCS are removed by GNG and the topology is determined during runtime.

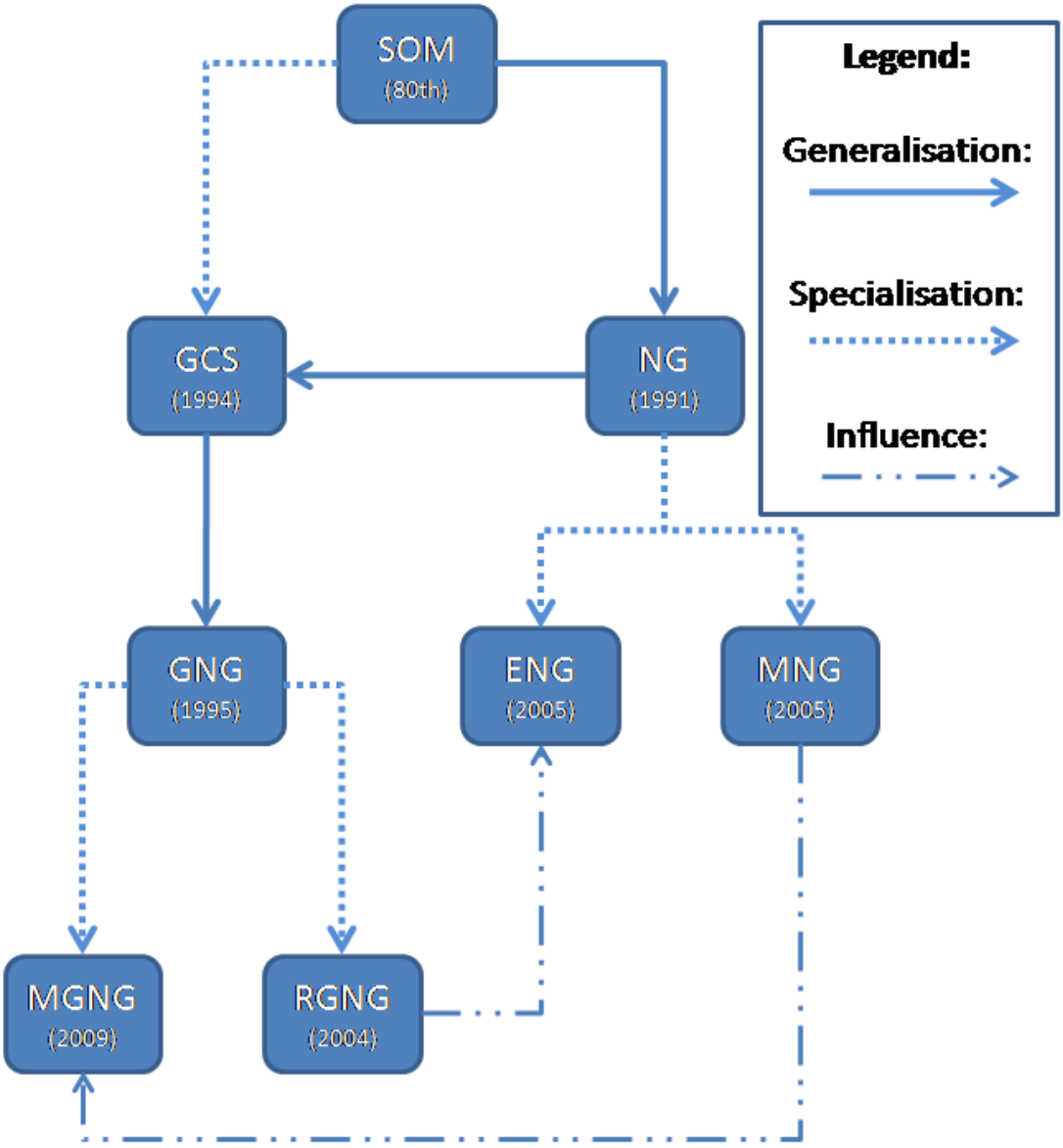


Figure 1: Functional/temporal dependency tree illustrating specialisation and generalisation among the proposed PBC-algorithm.

On basis of the GNG the *Robust Growing Neural Gas* (RGNG)-algorithm was proposed in 2004. RGNG uses the *Minimum Description Length* (MDL) for determination of the problems optimal network representation and further parameter to enhance GNG as specialisation and make it more robust towards noisy input data. The major parts of the RGNG algorithm were reused to enhance the NG algorithm in form of the *Enhanced Neural Gas* (ENG) in 2005. Therefore ENG is a specialisation of NG, strongly influenced by RGNG because the network has the capability of creating/deleting neurons. Beside ENG the *Merge Neural Gas* (MNG)-algorithm is also a specialisation of NG and was proposed in 2005. This algorithm is based on Merge-SOM (not presented in this thesis) and therefore connect the current input vector with previous ones to evaluate time series. The approach that previous input vectors influences the current input was adapted to GNG and results in the proposal of the *Merge Growing Neural Gas* (MGNG)-algorithm in 2009. Whereby MGNG is a specialisation of GNG, due to time series evaluation capabilities and a generalisation of MNG due to dynamic network size estimation.

# 4 Summary

This thesis has presented eight PBC-algorithms that are based on the SOM algorithm. It was ascertained that increasing the algorithms robustness leads to additional parameters (see section 2.2.1 and 2.4.1). Furthermore it was shown that time series evaluation requires an influence of previous input vectors towards the current input (see section 2.2.2 and 2.4.2). In summary it can be said that the GNG-algorithm is more flexible than NG and requires less parameters. For example, hierarchical GNG can be used in human action recognition ([PWW14]).

# List of Figures

# References

[AvHHB09] Andreas Andreakis, Nicolai von Hoyningen-Huene, and Michael Beetz. Incremental unsupervised time series analysis using merge growing neural gas. In José Carlos Príncipe and Risto Miikkulainen, editors, *WSOM*, volume 5629 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 2009.

[EZH+07] Pablo A. Estvez, Ricardo Zilleruelo-Ramos, Rodrigo HernÃ¡ndez, Leonardo Causa, and Claudio M. Held. Sleep Spindle Detection by Using Merge Neural Gas. In *6th International Workshop on Self-Organizing Maps (WSOM 2007)*, September 2007. URL: `http://liris.cnrs.fr/publis/?id=5580`, `doi:10.2390/biecoll-wsom2007-149`.

[EZRZ07] Pablo A. Estvez, Ricardo Zilleruelo-Ramos, and Jacek M. Zurada. Window Merge Neural Gas for Processing Pattern Sequences. In *IJCNN*, pages 2069–2074. IEEE, 2007. URL: `http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2007.html#EstevezZZ07`.

[Fri93] Bernd Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460, 1993.

[Fri95] Bernd Fritzke. A Growing Neural Gas Network Learns Topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[HA00] Victoria J. Hodge and James Austin. Hierarchical growing cell structures: TreeGCS. In Robert J. Howlett and Lakhmi C. Jain, editors, *KES*, pages 553–556. IEEE, 2000. URL: `http://dblp.uni-trier.de/db/conf/kes/kes2000.html#HodgeA00`.

[HMN+05] Barbara Hammer, Alessio Micheli, Nicolas Neubauer, Alessandro Sperduti, and Marc Strickert. Self-organizing maps for time series, 2005.

[MBS93] T. Martinetz, S. Berkovich, and K. Schulten. ¨"Neural-gas¨" Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE-Transactions on Neural Networks*, 4(4):558–569, 1993.

[MS91] T. Martinetz and K. Schulten. A ¨"Neural-Gas¨" Network Learns Topologies. *Artificial Neural Networks*, I:397–402, 1991.

[PWW14] German Ignacio Parisi, Cornelius Weber, and Stefan Wermter. Human action recognition with hierarchical growing neural gas learning. In *ICANN*, pages 89–96, 2014.

[QS04] A. Kai Qin and Ponnuthurai N. Suganthan. Robust growing neural gas algorithm with application in cluster analysis. *Neural Networks*, 17(8-9):1135–1148, 2004. URL: http://dblp.uni-trier.de/db/journals/nn/nn17.html#QinS04.

[QS05] A. K. Qin and P. N. Suganthan. Enhanced neural gas network for prototype-based clustering. *Pattern Recogn.*, 38(8):1275–1288, August 2005. URL: http://dx.doi.org/10.1016/j.patcog.2004.12.007, doi:10.1016/j.patcog.2004.12.007.